PharmaSUG 2024 - Paper AP- 212

R Shiny and SAS Integration: Execute SAS Procs from Shiny Application

Samiul Haque and Jim Box, SAS Institute

ABSTRACT

The integration of different programming languages and tools is pivotal for translational data science. R Shiny® is the most popular tool for building web applications in R. However, biostatisticians and data scientists often prefer to leverage SAS®. Procs or macros for clinical decision making. The world of R Shiny and SAS does not need to be decoupled. R Shiny applications can incorporate SAS procs and analytics. In this work, we present mechanisms for integrating R Shiny and SAS®. We demonstrate how SAS Procs and macros can be executed from R Shiny front end and SAS logs and results can be printed within Shiny App.

INTRODUCTION

R Shiny has emerged as a favored tool for developing dashboards and web applications within the life sciences industry. In teams with diverse programming backgrounds, there arises a need to execute validated SAS macros on study datasets and seamlessly incorporate the results into Shiny applications. For instance, users may wish to initiate an SDTM automation process, crafted with SAS Macros, directly from R Shiny applications. Another scenario involves generating ggplot2 figures based on outputs from a SAS macro. SAS Viya ® offers various avenues to establish connectivity between R and SAS sessions. In this paper, we demonstrate a method for integrating R Shiny applications with SAS, paving the way for seamless interaction between the two computes.

INTEGRATION MECHANISMS

Various methods exist for submitting SAS code from the R interface. Table 1 provides a summary of several available integration methods for SAS and R. All of these mechanisms are available as open source.

Integration Mechanism	Developed By	Link to Documentation
SWAT: SAS Wrapper for Analytics Transfers. SWAT allows R wrappers for connecting to SAS Viya in memory processing engine, also known as Cloud Analytics Services (CAS)	SAS	https://github.com/sassoftware/R-swat
REST API: SAS Viya provides public API endpoints to all modules. Public REST APIs can be used to execute and submit SAS programs, consume SAS resources, create, and manage SAS sessions from any external tools	SAS	https://developer.sas.com/guides/restapis/viya-rest.html

Integration Mechanism	Developed By	Link to Documentation
SASPy + Reticulate: SASPy is a python wrapper for executing SAS method from Python. SASPy works with SAS9 and Viya. Though this is a python wrapper it can be called from R using the **reticulate** package. Reticulate provides a mechanism for interoperability between R and Python.	SAS + Third Party [1]	https://support.sas.com/en/software/saspy.html https://rstudio.github.io/reticulate/index.html
SASR: SASR is a standalone open source package that provides interface to SAS through SASPy and Reticulate	Third Party [2]	https://github.com/insightsengineering/sasr

Table 1. Different mechanisms for integrating R and SAS

EXAMPLE – EXECUTING SAS PROCS FRM R SHINY APPLICATION

In this paper, we present an example of R Shiny and SAS integration, showcasing two methods outlined in Table 1: SWAT and SASR. We utilized the SWAT package to expose SAS in-memory tables to Shiny App users, enabling them to access live data from the SAS platform. Additionally, we utilized SASR to execute a proc on a selected dataset, retrieving the results and SAS log.

PREREQUISITES

The Shiny app was developed using the RStudio IDE, requiring the R-SWAT and sasr packages. For visualization, ggplot2 was utilized. sasr relies on SASPy for connecting to the SAS session, with modifications made to the SASPy configuration file to include the necessary connection strings. Detailed instructions can be accessed in the SASPy documentation (refer to Table 1). To connect to SAS Viya from SWAT, we provided the username and password through the R interface. For a secured connection to the SAS Viya platform, we installed the security certificate provided by the SAS Admin on our workstations.

SHINY APP FRONT-END

The Figure below shows the Shiny front end.

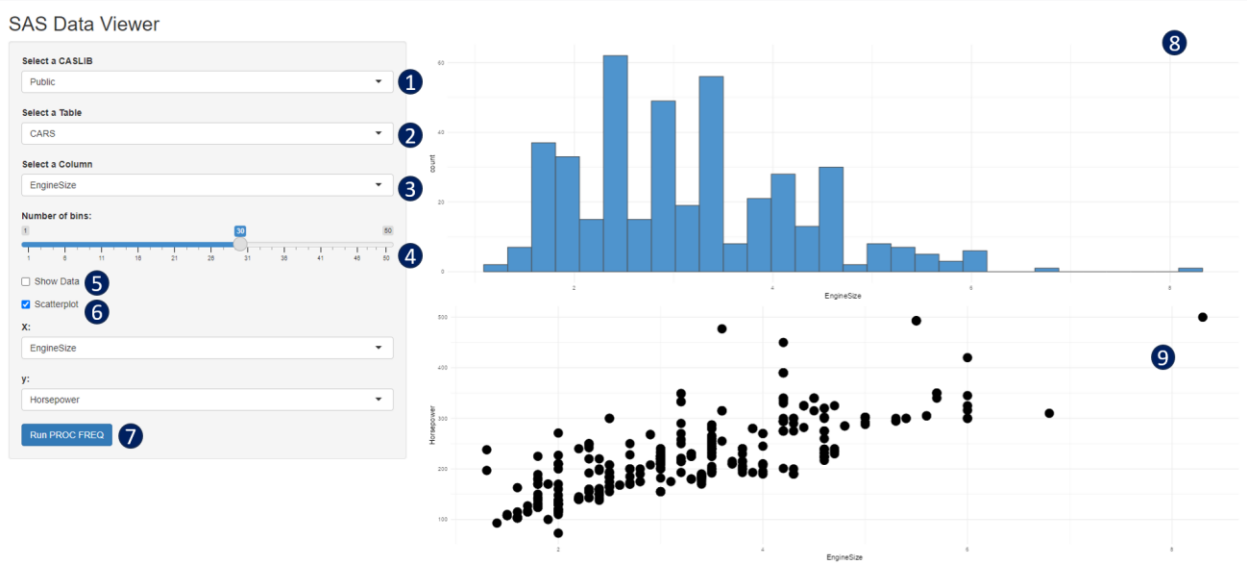

Figure 1. Shiny App Front-End

Different components in Figure 2 are numbered for clarity:

1. Dropdown element for selecting SAS in-memory library (CASLIB)
2. Table selector.
3. Column selector.
4. Slider: Adjusts the number of bins in the ggplot histogram (Circle 8).
5. Checkbox, Allows users to view detailed data.
6. Checkbox: Shows Scatterplot.
7. Circle: "Run PROC FREQ" button, enabling users to execute PROC FREQ on the selected table

PROC RESULTS AND LOGS

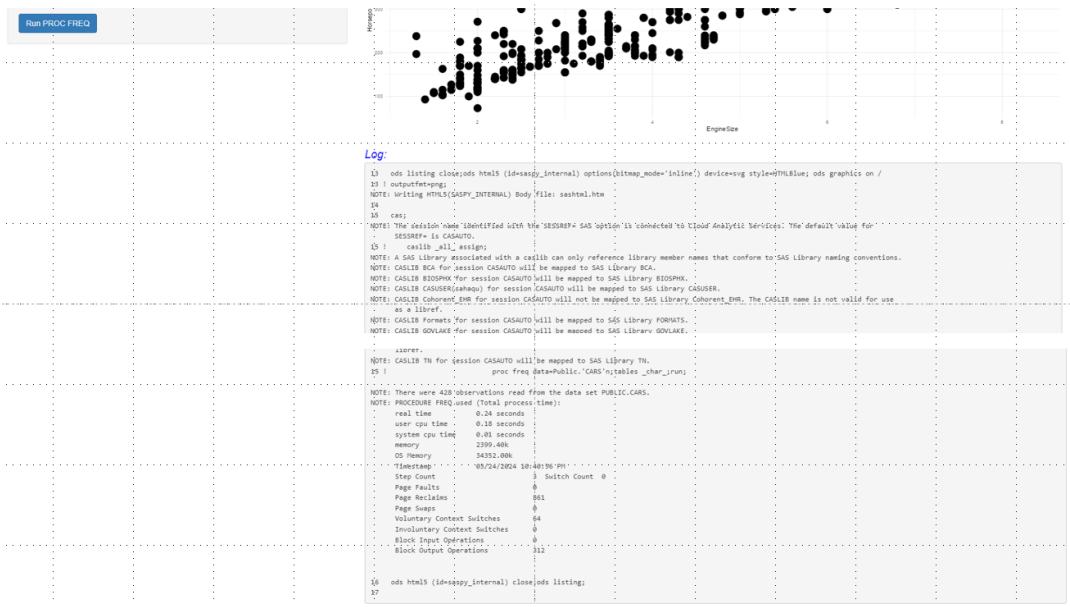

Figure 2. SAS Log is Printed on the Shiny Front-End after Completion of PROC

Result:

The SAS System

The FREQ Procedure

Make	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Acura	7	1.64	7	1.64
Audi	19	4.44	26	6.07
BMW	20	4.67	46	10.75
Buick	9	2.10	55	12.85
Cadillac	8	1.87	63	14.72
Chevrolet	27	6.31	90	21.03
Chrysler	15	3.50	105	24.53
Dodge	13	3.04	118	27.57
Ford	23	5.37	141	32.94
GMC	8	1.87	149	34.81

Figure 3 SAS System Output is Surfaced on the Shiny Front-End

CODE

The complete codebase of the application is available on GitHub at the following link: https://github.com/samiulhq/RShiny_SASViya While we provide some example snippets in this paper for understanding the overall integration mechanisms, we highly encourage readers to refer to the GitHub repository for implementing their own use cases..

Establishing a SAS Session

The following R code snippet demonstrates how to establish a connection to a SAS compute session:

```
library(sasr)
my_sas_session <- get_sas_session()
```

Executing SAS Code

The following R code snippet demonstrates how to execute SAS code in the SAS compute session. The resulting object is a list that contains the log and result of the run.

```
result <- run_sas("
   proc freq data = sashelp.heart;
   tables sex / out=FreqCount;
   run;
 ")#See Log
cat(result$LOG)

#See Result
 cat(result$LST)
```

Fetching data from SAS session

We can retrieve a table from the SAS session into R as a dataframe. In the following example, we fetch the FreqCount table from the work library into the R session.

```
freq_count <-  my_sas_session$sd2df('FreqCount',libref='WORK')
 print(freq_count)
```

Uploading data to SAS session

Data can be uploaded to SAS from R session using the following method:

```
upload <- my_sas_session$df2sd(mtcars,table ='mt_df',libref='WORK')
```

List available tables in a SAS Library

```
my_sas_session$datasets(libref="WORK")
```

End a SAS Compute Session

```
my_sas_session$endsas()
```

Connection to CAS

Cloud Analytics Services (CAS) is the in-memory high-performance engine on SAS Viya. You can establish a connection to CAS from your R session using the SWAT package:

```
cassess<-CAS("<url to CAS server>",<port
number>,username=<userid>,password=<password>)
```

using the CAS session, all in memory operations and actions can be performed from the R native interface. The following examples demonstrates how to list all in-memory libraries in the cassess object available to the user:

```
list_caslibs<-cas.table.caslibInfo(cassess)
```

CONCLUSION

In conclusion, we have demonstrated an easily implementable approach for integrating R and SAS. Leveraging a combination of SWAT, SASR, and SASPy, developers can create sophisticated web applications on a validated SAS Viya platform. This integration facilitates seamless collaboration between R and SAS environments, opening up new possibilities for data-driven applications in translational data science.

REFERENCES

Ushey K, Allaire J, Tang Y (2024). reticulate: Interface to 'Python'. R package version 1.35.0, https://github.com/rstudio/reticulate.

Roche./Genentech – Insights Engineering sasr , https://github.com/insightsengineering/sasr

SAS Software R-swat, https://github.com/sassoftware/R-swat

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Samiul Haque	Jim Box
Samiul.Haque@sas.com	Jim.Box@sas.com
https://www.linkedin.com/in/samiulhaque/	https://www.linkedin.com/in/jwbox/